Comparison of Four Randomized Optimization Algorithms

Yurong Fan
Graduate Student, College of Computing, Georgia Tech

INTRODUCTION

Optimization is a crucial component in machine learning. This paper explores the performance of 4 randomized optimization algorithms – Randomized Hill Climbing (RHC), Simulated Annealing (SA), Genetic Algorithm (GA), and MIMIC with empirical evidences on 3 discrete parameter space problems – N Queen, Travelling Salesperson Problem, and Max K Coloring, and 2 continuous parameter space problems – optimizing the weights of a neural network of 1) a digital recognition dataset and 2) the Iris dataset, aiming to answer the question – which randomized optimization algorithm is more suitable for which circumstances.

METHOD

I. Implementation of the Randomized Optimization Algorithms

The three optimization algorithms are implemented with the mlrose library (https://mlrose.readthedocs.io/en/stable/) in Python written by Genevieve Hayes. I adapted this implementation a little bit to get more outputs from it. The adapted mlrose and Python scripts wrote to conduct experiments and analysis can be found on my Github.

II. Evaluation of the Algorithm Performance

1. Fitness

An optimization problem is defined by Russell and Norvig (2010) as a problem in which “the aim is to find the best state according to an objective function”. The “best” optimization algorithm finds the state that maximizing a fitness function (or minimizing a loss function). Thus the fitness/loss achieved by an algorithm is used to evaluate its performance. Please note that in most problems examined in this paper, there are multiple local optima instead of one global optima. It’s not feasible to conclude whether an algorithm reached the global optima from its fitness value. But it’s easy to tell that an algorithm stuck in local optima if other algorithm achieved better fitness in the same circumstance. 3 metrics are used to evaluate fitness:

1) Average fitness: average fitness value achieved by the algorithm in each convergence. The interpretation of the value depends on specific fitness function of each problem. For

example, in the neural network problem, it’s log loss (as it’s a classification problem). In the N-Queen problem, it’s the number of pairs of attacking queens.

2) # Best fitness: number of times/convergence an algorithm achieved the best fitness among all algorithms.
3) # Worst fitness: number of times/convergence an algorithm achieved the worst fitness among all algorithms.

2. Convergence Time

The amount of time used by an algorithm to converge is another consideration for evaluating the performance of a randomized optimization algorithm. 2 metrics are used to evaluate the time efficiency:

1) Average wall clock time spend till converge
2) Average iterations till converge: each time an algorithm evaluates the fitness of a new neighbor or new population for genetic algorithm is considered one iteration.

3. Model Accuracy

In the neural network weight optimization problem, machine learning model accuracy measures were used to compare across randomized optimization algorithms and compare with Backpropagation. As the problem the neural network was tackling is multi-class classification, macro F1 score is the metric chosen to evaluate model accuracy. And hold-out validation is used to get F1 score on both the training set and the test set.

III. Description of the Experiments

Many factors count toward the performance discussed in the above sector. In order to make fair comparison of the performance between the randomized optimization algorithms, other factors should be well controlled or at least be explicit to aid fair conclusion. The three biggest factors that influence optimization performance and addressed in the experiments include:

1) **The problem**: the complexity of the problem
2) **The algorithm**: the implementation and hyper parameter setting of each algorithm
3) **Randomness**: The randomness associated with the randomized optimization

To address factor 1), all optimization algorithms were applied on the exact same problem each time. Specifically,

- **In the N-Queen problem**, problems of different number of Queens were examined to explore this type of problem but with different level of complexity. And comparison across algorithms (# best fitness, # worst fitness metrics) were made at each level of complexity.

- **In the TSP problem**, I didn't explore different number of cities (can be a further work), but fixed the number of cities to a typical number usually used for this problem – 8. Though different city coordinates were generated randomly (discussed more in factor 3) randomness), comparison were made in the same city coordinates first.
- **In the Max K Coloring problem**, I didn't explore different number of cities, but fixed the number of nodes in the graph to 20, the number of edges to 60, and the number of color to 3. These special settings do limit the generalization of the related conclusions reached from this study. And further work can be done to explore different complexity.
- **In the Neural Network problem**, I fixed the neural network to have 1 hidden layer and 20 nodes in the hidden layer for both problems considering the long runtime. Though it is not the optimal structure for the best generalization accuracy.

To address factor 2), the same implementation and hyper parameter setting of each optimization algorithm were used across the 3 discrete parameter space problems and the 1 neural network problem. According to the No Free Lunch Theorem, there is no optimal parameter configuration for all problems. Literatures were reviewed to find "common" settings. And settings that can strengthen an algorithms advantage were leaned toward. Below are highlights of the algorithm implementations.

- **Randomized Hill Climbing (RHC):** two versions were implemented - one without random restart (named "RHC wo restart"), and one with 5 random restarts (named "RHC 5 restart") to examine the performance improvement from random restarts. In theory, random restart can help escape from local optima and increase the chance of reaching global optima. But more random restarts also means longer convergence time.
- **Simulated Annealing (SA):** unlike RHC that employed random restart to escape local optima and explore wider spaces, SA simulated the cooling of material in a heat bath, and the exploration is done at high temperatures. Different methods to set the temperature were suggested in literatures. The maximum fitness distance between one neighbor and another can be used as a reference to set the starting temperature. Rayward-Smith (1996) suggested to start with a very high temperature and cool down rapidly until about 60% of worst solutions are being accepted. There is a trade-off between the cool down time – related to the training time, and the "exploration" effort of the algorithm. I want to strengthen the "exploration" ability of the Simulated Annealing and thus selected a conservative temperature structure – high starting temperature (10000) and slow decay rate (0.95) following geometric decay. The convergence time of SA should be exaggerated as a result.
- **Genetic Algorithm (GA):** the exploration of GA coming from the crossover and the mutation process, and the exploit of GA is done through the "survival" of a subset of the population. Related hyper parameters include population size, mutation rate, the form of crossover. In my experiments, a set of "common" parameters were used, which is 200 population size, 10 percent mutation rate, and one-point cross-over. Please also note the trade-off between convergence time and the exploration ability of the algorithm.

Specifically, raising the population size and/or raising the mutation rate will slow convergence.

- **MIMIC:** a common set of parameters were used including 200 population size, and 20 percent of samples to keep at each iteration.
- **Stopping criteria:** I set maximum attempt (number of attempt to make before stop if no neighbor/population has fitness increase) to 10 for RHC/SA/MIMIC on the discrete parameter problems, and 50 for GA as I found GA is usually not "lucky" enough within 10 generations. In neural network weights optimization problems, algorithms stop when they reach a specified iteration number.

To address factor 3), each type of problem was solved by the same algorithm for a meaningful times of convergence / "trials". The performance gain or loss due to a good or bad initial random state or other randomness (such as the random choice of neighbors) should be "averaged out".

- **In the N-Queen problem:** 40 trials were performed by each algorithms on N-Queen problems with queen number of board size ranged from 4 to 200.
- **In the TSP problem:** 100 sets of 8-city coordinates were randomly generated (following a uniform distribution within a 100 * 100 map). And all algorithms were applied on these 100 problems in 100 trials.
- **In the Max K Coloring problem:** 100 graphs were generated randomly. And all algorithms were applied on these 100 problem in 100 trials.

EXPERIMENT RESULTS

1. Discrete Parameter Space Problems

A. The N-Queen Problem

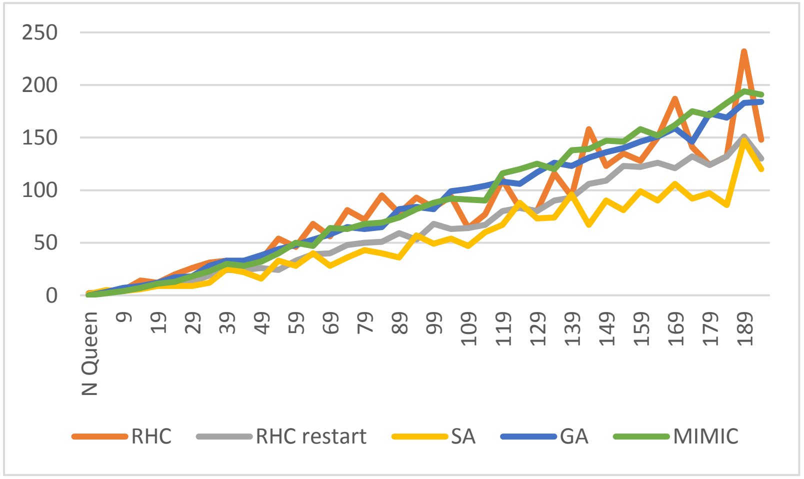


Figure 1: N-Queen by algorithm best Loss at different number of queens

40 N-Queen Trials with different N	# Best Fitness	# Worst Fitness	Avg Loss	Worst Loss	Avg wall clock time (seconds)	Avg iterations
HC (wo restart)	1	16	82.7	232	2.1	76.9
RHC (w 5 restarts)	7	0	65.2	151	11.4	437.3
SA	32	1	53.6	147	32.2	1116.2
GA	1	11	86.1	184	61.8	15.8
MIMIC	3	14	88.1	194	185.5	13.1

Figure 2: 40 N-Queen Trials – Algorithm Performance Summary

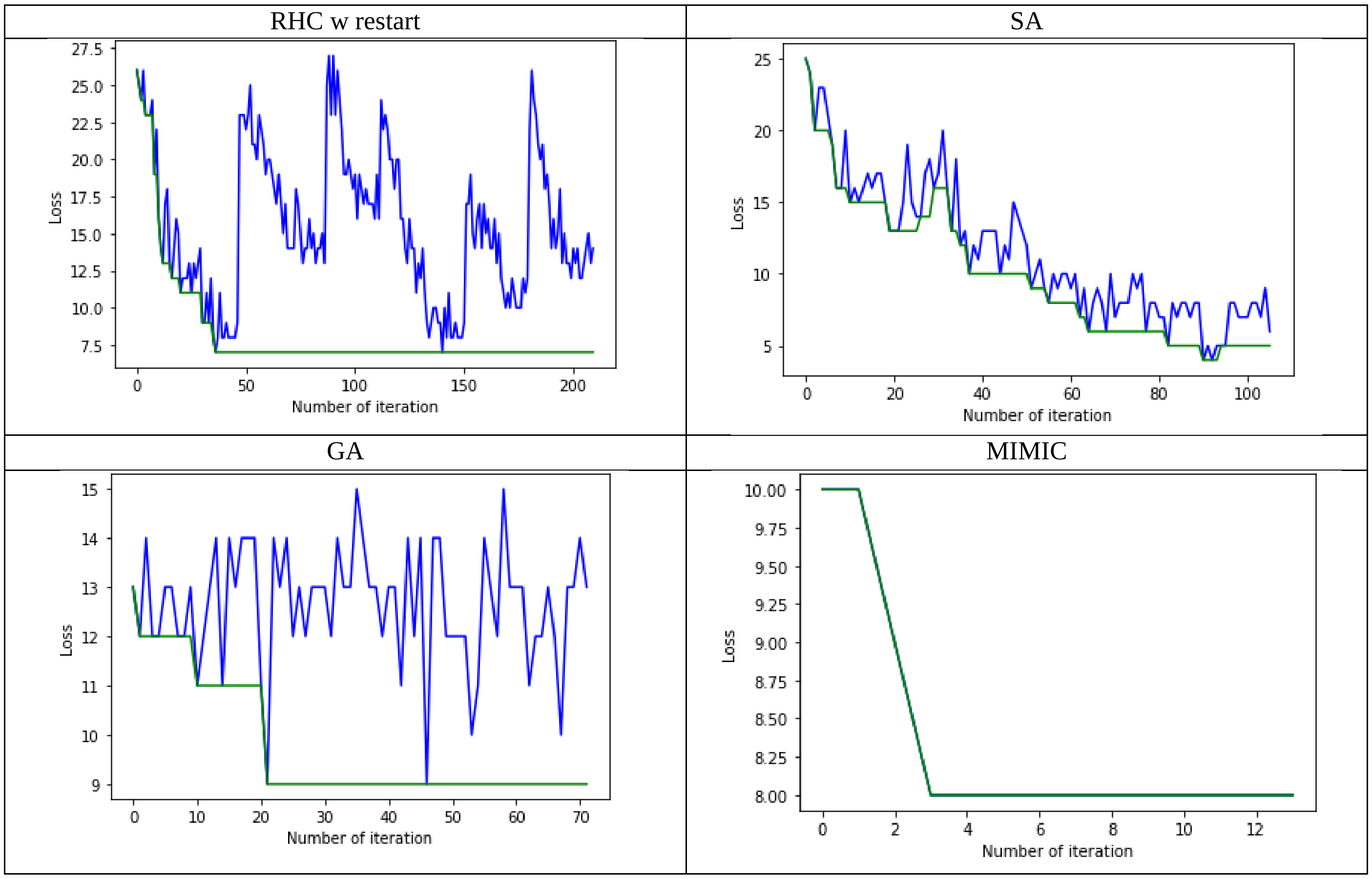


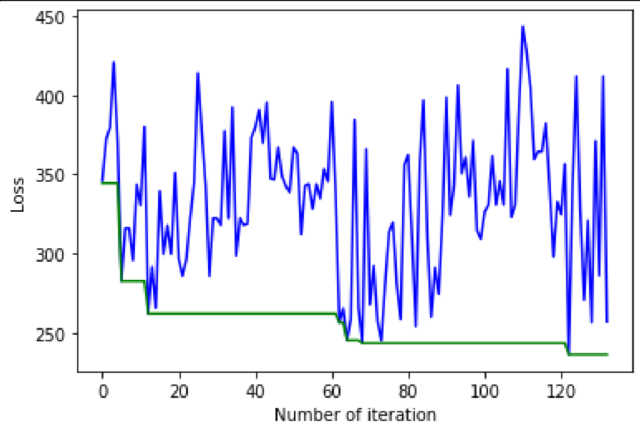
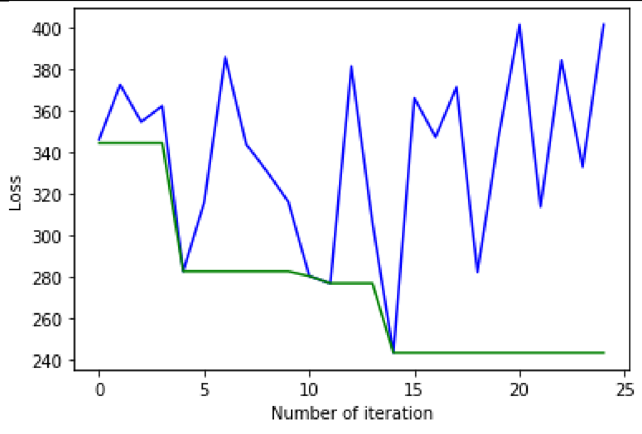
Figure 3: One 20-Queen Trial - Loss Curve over Iterations
(blue line: evaluated neighborhood/population loss; green line: best loss)

SA performs best in the N-Queen problem in terms of fitness/loss. In Figure 1, SA consistently excelled in the N-Queen problem at different number of queens, and the advantage of SA over other algorithms enlarged as the number of queens increased. From Figure 2, SA achieved the best fitness in 32 out of 40 trials, which is a clear win. I think the reason is that in the N-Queen problem, we can reducing the number of attacking queen pairs by making local move without considering past moves. Local search algorithms HC/RHC/SA is suitable for these type of problems. The challenge is local optima as we can see in Figure 2 that HC stuck in local optima with poor fitness. RHC/SA which can escape from local optima and try to search the global optima achieved the best fitness. Comparing RHC and SA in Figure 3, RHC was not lucky enough and always ended up in some local optima in its 5 + 1 hill climbing exercises from the blue line of RHC in Figure 3. There are more local optima in N-Queen as N increases. And when having a slow temperature cool down schedule (see III. Description of the Experiments, factor 2) for detailed parameter setting), SA can get through valleys (accepting many higher loss neighbors as shown in the blue line of SA in Figure 3) to explore a much wider solution space. And the role of "luck" for SA is not as big as it is for RHC. MIMIC and GA had the worst performance. We can see from the blue lines in Figure 3, the fitness change over generations is abrupt for GA, and MIMIC only made one improvement. Clearly the smooth paths in the "hills and valleys" of N-Queen fitness space were not utilized by GA and MIMIC.

B. Travelling Salesperson Problem

100 TSP Trials	# Best Fitness	# Worst Fitness	Avg Loss	Worst Loss	Avg wall clock time (seconds)	Avg iterations
HC (wo restart)	5	69	312.0	482.8	0.0087	24.3
RHC (w 5 restarts)	49	5	275.2	370.7	0.0482	146.7
SA	21	30	291.2	395.2	0.0103	151.5
GA	80	1	269.9	342.3	8.4295	83.2
MIMIC	62	4	272.1	342.3	3.6660	11.4

Figure 4: 100 Travelling Salesperson Problem Trials - Algorithm Performance Summary

RHC w restart	**SA**
	
GA	**MIMIC**

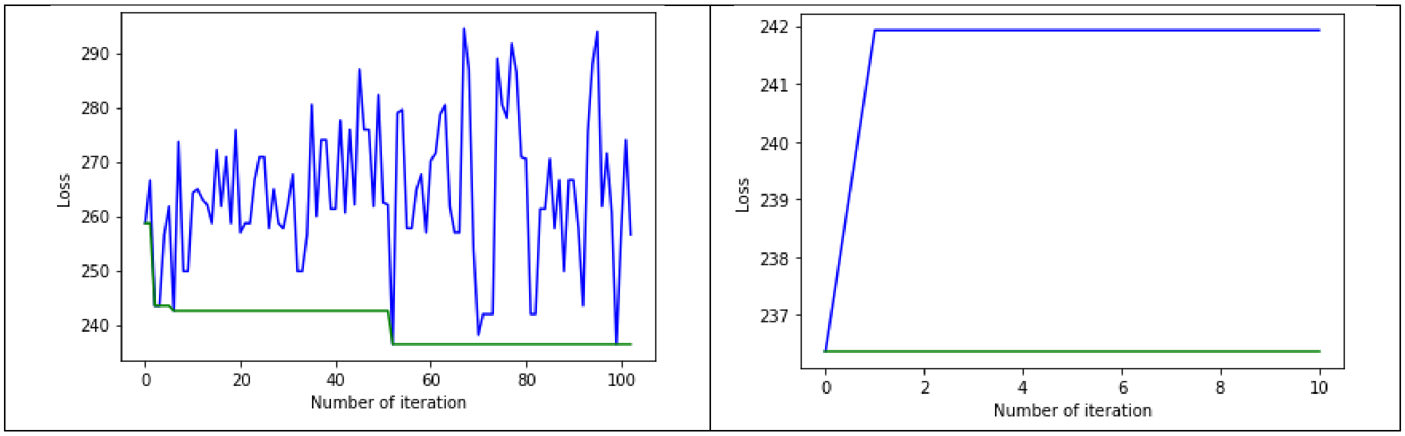


Figure 5: One TSP Trial - Loss Curve over Iterations
(blue line: evaluated neighborhood/population loss; green line: best loss)

GA outperformed in TSP. As shown in Figure 4, GA got the best fitness in 80 out of the 100 trials with the lowest average and worst loss. TSP is a NP-hard problem. The visiting orders of each city are interconnected, and we cannot only change the order of one city to reduce the travel distance. Thus RHC/SA failed as they cannot improve fitness through making local change. The underline distribution of the fitness function is also hard to be modeled by MIMIC. GA doesn't rely on making local search (though offspring would be "close" to parents) or modeling underline distribution in optimization. It's especially suited to tasks in which hypotheses are complex, and in which the objective to be optimized may be an indirect function of the hypothesis. One caveat is that the wall clock runtime of GA was longer than the other algorithms from Figure 4. Actually its runtime is of different order of magnitude comparing with the runtime of RHC and SA. Though GA didn't make many iterations (tunable by the stopping criteria, see III. Description of the Experiments, factor 2), it took longer time in each iteration for its crossover, mutation, and evaluating the fitness of the population.

C. Max K Coloring

100 MaxKColor Trials	# Best Fitness	# Worst Fitness	Avg Loss	Worst Loss	Avg wall clock time (seconds)	Avg iterations
HC (wo restart)	1	86	12.2	18	0.0025	24.6
RHC (w 5 restarts)	12	12	8.2	11	0.0142	161.8
SA	17	11	8.2	14	0.0119	242.1
GA	3	13	8.7	10	3.4971	74.4
MIMIC	90	0	5.7	8	5.0548	16.3

Figure 6: 100 Max K Coloring Trials - Algorithm Performance Summary

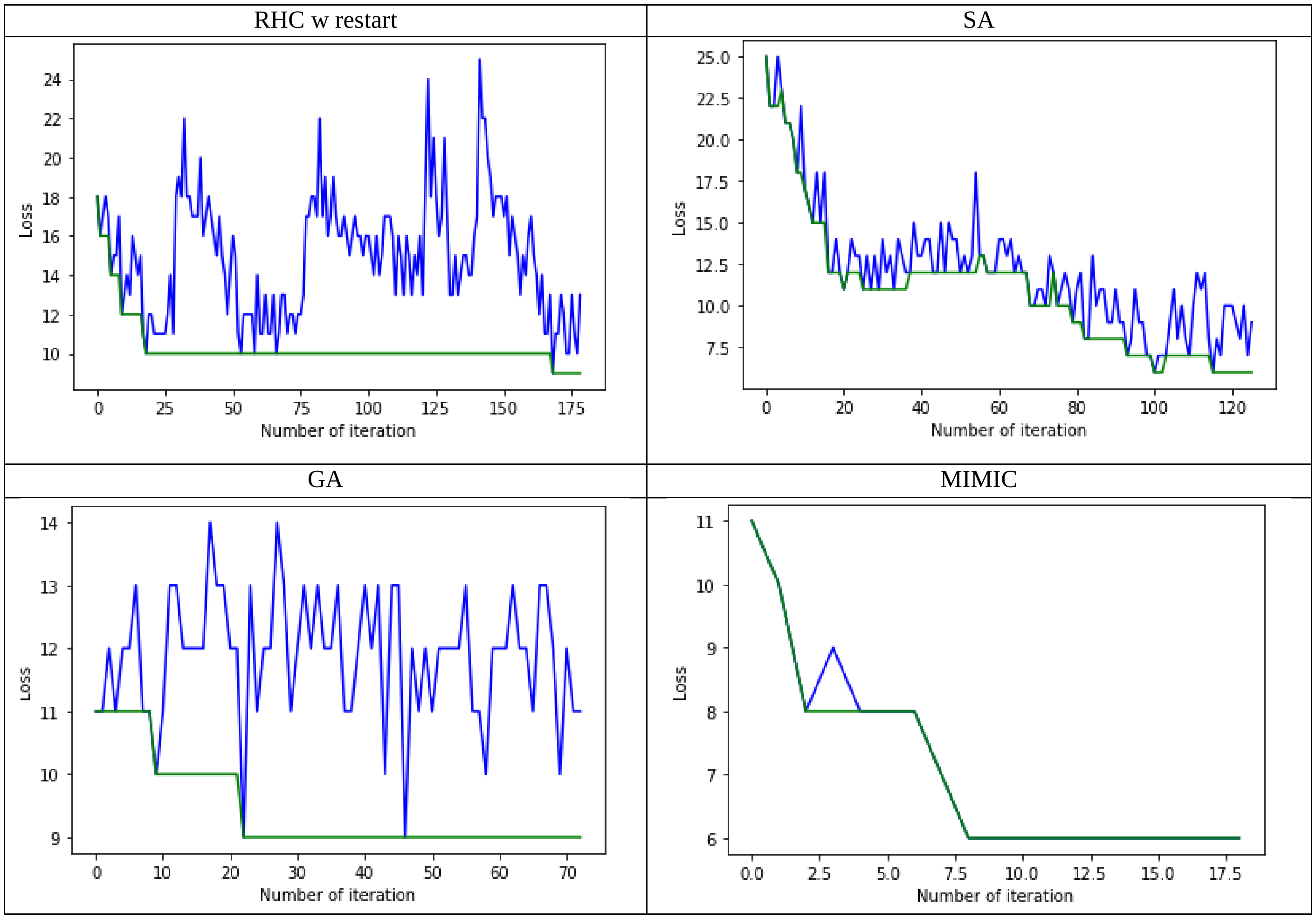


Figure 7: One Max K Coloring Trial - Loss Curve over Iterations
(blue line: evaluated neighborhood/population loss; green line: best loss)

MIMIC outperform other algorithms in 90 out of 100 trials in Max K Coloring regarding the fitness. Max K Coloring problem is known to be a NP-Complete problem. The coloring of each node is dependent upon each other to minimize the loss. It is not the single values in the parameter space, but rather the structures of the parameters matter. MIMIC is the only algorithm among the four that analyze the global structure of the optimization space. From the blue line of MIMIC in Figure 7, MIMIC didn't use many iterations to reach an ideal fitness compared with the rest algorithms, it actually made a few big improvement. The drawback of MIMIC is that it took longer for each iteration as it does more calculation (such as inferring the underline distribution of fitness). GA performed worst in this problem. Its 1-point crossover can possibly "break up" a good coloring structure. RHC/SA reduced the loss function substantially, but not as much as MIMIC. I think it's because that local search can make some improvement in Max K Coloring problem in some situations. For example, RHC/SA can improve fitness when simply change the coloring of one node can reduce the number of edges connecting nodes of the same color. After there is no such node, RHC/SA cannot make further improvement. The remaining better solutions are not reachable through local search, but can be discovered by MIMIC as a whole.

2. Neural Network Weight Optimization

A. Digit Recognition Dataset

	# Iterations	Runtime (seconds)	Runtime/Iteration (seconds)	Starting/End Loss	Train/Test F1 score
RHC	10000	6382	0.638	4.010/2.967	0.174/0.170
SA	8000	7824	0.978	4.010/3.785	0.1011/0.1012
GA	100	9931	99.310	4.010/3.571	0.119/0.116

Figure 8: Digit Recognition Neural Network – Algorithm Performance Summary

The digit recognition dataset from assignment 1 can be downloaded at https://www.kaggle.com/c/digit-recognizer. Its input features 784 pixel values of handwritten images of digits, and its target values are 10 digits.

The biggest learning from applying randomized optimization algorithms on this dataset is that none of these algorithms can beat backpropagation in terms of the convergence speed. This neural network has 15600 weights coupling with the 21000 rows of training data. This number of weights and the size of the training data are actually common in modern neural network training. As shown in Figure 8, all three algorithms were far from convergence after a runtime of 6000 – 10000 seconds for each algorithm.

To understand why, the runtime/iteration of backpropagation on this problem is 0.900 second, which is similar to the runtime/iteration of RHC and SA. However, RHC and SA at most update 1 weight in each iteration, while backpropagation updates all 15600 weights in each iteration. Secondly, backpropagation uses gradient decent to guide search toward the direction of the steepest fitness change, while RHC/SA make random search toward all possible directions, and GA make random crossover/mutation for exploration. Gradient decent makes the best use of the continuous and differentiable hypothesis space of neural network weights.

B. Iris Dataset

	# Iterations	Runtime (seconds)	Loss	Train/Test F1 score
Backpropagation	200	0.10	n.a.	1.0/0.9564
RHC	10000	23.73	0.9850/0.0119	1.0/0.9564
SA	10000	35.42	0.9852/0.1347	0.9340/0.8857
GA	10000	3924.88	0.5026/0.0136	1.0/0.9564

Figure 9: Iris Classification Neural Network – Algorithm Performance Summary

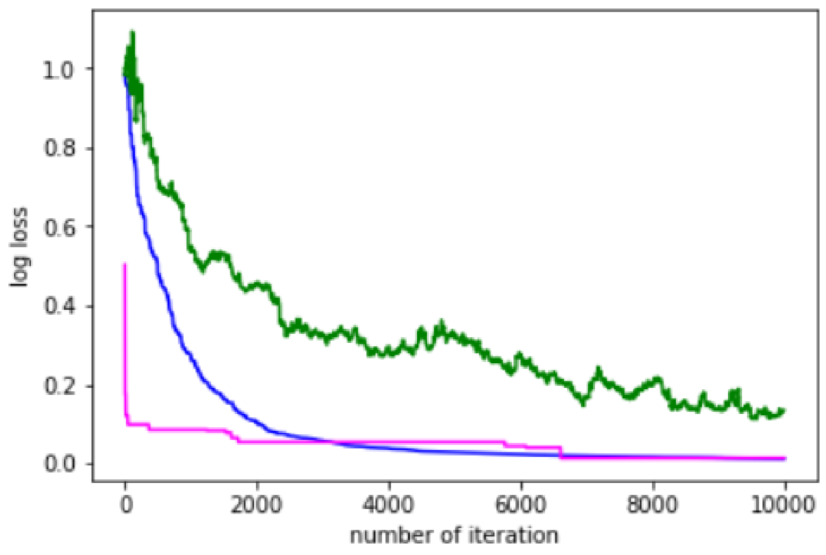


Figure 10: Iris Classification Neural Network – Loss over Iterations (green line: SA, blue line: RHC, pink line: GA)

To further explore, I applied the algorithms on neural network of the Iris dataset. This dataset is way smaller than the digit recognition dataset with 120 rows in the training set (80:20 train/test split ratio), 4 input features. With 20 nodes in the hidden layer, the network ends up with 160 weights. Now the runtime of all three algorithms is acceptable for my local machine, and I set their iterations to 10000 for easy compare across algorithms. We can see from Figure 9 that with sufficient iterations, RHC and GA converged to the same loss and F1 scores of backpropagation. SA is still “wondering” when it reached 10000 iterations (further work can be done to explore if it’s due to the temperature schedule). As discussed in the above sector, RHC and gradient decent are actually both hill climbing algorithms. I think the biggest difference is that gradient decent utilized the differentiable property of continuous hypothesis space to make the climbing more efficient. Thus it’s not surprising to the see the performance of RHC given enough iterations. Time complexity of GA observed in this and all other experiments are still its biggest drawback.

REFERENCE

1) Tom Mitchell. Machine Learning.

2) De Bonet, J. S., Isbell Jr, C. L., & Viola, P. A. (1997). MIMIC: Finding optima by estimating probability densities. In Advances in neural information processing systems (pp. 424-430).

3) Shmoys, D. B. (1985). The traveling salesman problem: a guided tour of combinatorial optimization (Vol. 3, pp. 1-463). E. L. Lawler, J. K. Lenstra, & A. R. Kan (Eds.). New York: Wiley.

4) R. L. Haupt, "Optimum population size and mutation rate for a simple real genetic algorithm that optimizes array factors," IEEE Antennas and Propagation Society International Symposium. Transmitting Waves of Progress to the Next Millennium. 2000 Digest. Held in conjunction with: USNC/URSI National Radio Science Meeting (C, Salt Lake City, UT, 2000, pp. 1034-1037 vol.2.

5) Wolpert, D. H., & Macready, W. G. (1997). No free lunch theorems for optimization. IEEE transactions on evolutionary computation, 1(1), 67-82.

6) Chalup, Stephan & Maire, Frederic. (1999). A study on hill climbing algorithms for neural network training. Proceedings of the 1999 Congress on Evolutionary Computation, CEC 1999. 3. 2021 Vol. 3. 10.1109/CEC.1999.785522.